# Probabilistic System-on-a-Chip Architectures

LAKSHMI N. CHAKRAPANI, PINAR KORKMAZ, BILGE E. S. AKGUL, and KRISHNA V. PALEM
Georgia Institute of Technology

Parameter variations, noise susceptibility, and increasing energy dissipation of CMOS devices have been recognized as major challenges in circuit and microarchitecture design in the nanometer regime. Among these, parameter variations and noise susceptibility are increasingly causing CMOS devices to behave in an "unreliable" or "probabilistic" manner. To address these challenges, a shift in design paradigm from current-day deterministic designs to "statistical" or "probabilistic" designs is deemed inevitable. To respond to this need, in this article, we introduce and study an entirely novel family of probabilistic architectures: the *probabilistic system-on-a-chip* (PSOC). PSOC architectures are based on CMOS devices rendered probabilistic due to noise, referred to as *probabilistic* CMOS or PCMOS devices. We demonstrate that in addition to harnessing the probabilistic behavior of PCMOS devices, PSOC architectures yield significant improvements, both in energy consumed as well as performance in the context of probabilistic or randomized applications with broad utility. All of our application and architectural savings are quantified using the product of the energy and performance, denoted (energy $\times$ performance): The PCMOS-based gains are as high as a substantial multiplicative factor of over 560 when compared to a competing energy-efficient CMOS-based realization. Our architectural design is application specific and involves navigating design space spanning the algorithm (application), its architecture (PSOC), and the probabilistic technology (PCMOS).

Categories and Subject Descriptors: C.3 [**Computer Systems Organization**]: Special-Purpose and Application-Based Systems; C.1.3 [**Processor Architectures**]: Other Architecture Styles

General Terms: Performance, Design, Reliability

Additional Key Words and Phrases: Embedded systems, probabilistic computing

**ACM Reference Format:**

Chakrapani, L. N., Korkmaz, P., Akgul, B. E. S., and Palem, K. V. 2007. Probabilistic system-on-a-chip architectures. ACM Trans. Des. Autom. Electron. Syst. 12, 3, Article 29 (August 2007), 28 pages. DOI = 10.1145/1255456.1255466 http://doi.acm.org/10.1145/1255456.1255466

This work was supported in part by DARPA under seedling Contract F30602-02-2-0124, the DARPA ACIP Program under Contract FA8650-04-C-7126 through a subcontract from USC-ISI, and by an award from Intel Corporation.

Authors' addresses: L. N. Chakrapani (contact author), P. Korkmaz, B. E. S. Akgul, K. V. Palem, Center for Research on Embedded Systems and Technology, Georgia Institute of Technology, Atlanta, GA 30332; email: lnc@gatech.edu.

## 1. INTRODUCTION

Sustaining Moore's law [Intel 2007] through technology scaling into the nanometer regime poses several challenges. Chief among these are the hurdles introduced by noise, parameter variations, and other device perturbations [Natori and Sano 1998; Sano 2000; Kish 2002]. Such "statistical" or "probabilistic" behavior is inevitable at the device level of individual transistors, and current-day techniques of addressing these challenges, mainly in the form of rigorous test methodologies, are unlikely to be adequate for future technology generations. To accommodate and remedy this statistical behavior at the device level, it has been speculated that a shift in design paradigm—from current-day deterministic designs to statistical or probabilistic designs—would be necessary [Borkar et al. 2003]. Quoting the ITRS road map, "[r]elaxing the requirement of 100% correctness for devices and interconnects may dramatically reduce costs of manufacturing, verification, and test. Such a paradigm shift is likely forced in any case by technology scaling, which leads to more transient and permanent failures of signals, logic values, devices, and interconnects" [ITRS 2002].

To respond to this critical need, we introduce and study an innovative approach towards error tolerance and statistical designs in this article: a novel family of *probabilistic* architectures which we refer to as probabilistic *system-on-a-chip* (or PSOC). In our current context, these PSOC architectures are based on CMOS devices whose behavior is rendered probabilistic by noise. Such PSOC architectures not only allow us to tolerate device-level perturbations due to noise, but also harness noise to perform useful computations.

The surprising fact that "noise" can be harnessed for performing computation had been demonstrated earlier using abstract computational models [Palem 2003a, 2003b]. Building on these computational models and by using classical thermodynamics, it has been shown that energy savings can be achieved while performing meaningful computation by harnessing noise [Palem 2005]. We have validated these results (derived in the domain of computational models using classical thermodynamics) in the domain of CMOS, by innovating and studying CMOS devices whose behavior is rendered probabilistic due to ambient thermal noise [Cheemalavagu et al. 2004; Korkmaz et al. 2006]. We referred to such "noisy" or "unstable" CMOS devices as probabilistic CMOS or PCMOS devices.

In this article, we demonstrate that PCMOS technology not only yields energy savings at the device level, but also yields significant energy and performance benefits at the application level. To reiterate, we demonstrate that statistical behavior at the device level can be tolerated and harnessed to realize low-energy and high-performance computation. We accomplish this by designing and analyzing PSOC architectures which use PCMOS technology for a set of applications based on probabilistic algorithms. In addition, since design considerations for PSOC architectures differ significantly from those of conventional (deterministic) system-on-a-chip (SOC) architectures, we introduce and employ a novel algorithm-architecture-technology ($A^2T$) codesign methodology to design efficient PSOC implementations.

The rest of the article is organized as follows. In Section 2, we provide a historical perspective of the underlying foundational principles of PCMOS devices. We also summarize related work, which includes a summary of theoretical and engineering aspects of designs which tolerate statistical behavior of CMOS logic primitives. In Section 3, we describe probabilistic system-on-a-chip architectures and the central idea behind their implementation. For completeness, we briefly discuss PCMOS technology in Section 3.1. The energy and performance modeling methodology which we adopt to evaluate PSOC architectures is described in Section 3.3. We then describe our metrics (Section 3.4), which we use to study the performance of PSOC architectures. As mentioned earlier, our PSOC codesign methodology (the ($A^2T$) codesign methodology) differs from conventional codesign methodologies and is central to achieving the energy and performance benefits reported in this article. This codesign methodology, the main technology and algorithm characteristics which influence this methodology, and the application characteristics of PSOC detailed designs are detailed in Section 4. We present results (Section 4.2), and analyze them to account for and explain the energy and performance gains observed in PSOC implementations in Section 4.3. We discuss application optimization and PSOC implementation in detail in Section 5. Finally, we conclude and discuss future research directions in Section 6.

## 2. HISTORICAL PERSPECTIVE AND RELATED WORK

The connection between energy and computation can be found explicitly in Landauer's explanation of the paradox of Maxwell's demon. Maxwell's demon [Leff and Rex 1990] is a thought experiment first proposed by Maxwell, where an intelligent being seemingly violates the second law of thermodynamics by performing operations at a molecular level. Landauer demonstrated that any mechanical Maxwell's demon operating under a closed thermodynamic cycle would dissipate energy, and hence not violate the second law of thermodynamics [Landauer 1961]. He went on to conjecture that, by extension, any logical step of computation would consume $kT \ln 2$ joules of energy. Bennett's significant extension of Landauer's work showed that only *irreversible* or *nonrecovering* models of computing (where energy, once expended, cannot be or is not recovered) are subject to this limit (sometimes referred to as Landauer's limit) and that all computation can, in theory, be performed reversibly [Bennett 1973] with arbitrarily low energy consumption. A further refinement to Landauer's limit was proposed by Palem [2005], where it was shown that the energy necessary to compute a bit of information is related to its probability of correctness $p$. Meindl and Davis showed that Landauer's fundamental limit is applicable in the domain of CMOS devices (which perform irreversible switching) [Meindl and Davis 2000]. An extension of Palem's and Meindl and Davis's results showed that such a relationship between energy and probability of correctness holds in the domain of CMOS devices as well [Cheemalavagu et al. 2005]. These insights and results form the basis of energy-efficient probabilistic computing using PCMOS devices.

Utilizing defective logic elements for reliable computing is a concept which dates back to von Neumann's seminal work, where he studied techniques like

NAND multiplexing and majority voting to increase reliability of faulty logic gates [von Neumann 1956]. The author showed that if the failure probability of gates is statistically independent and low, computation can be performed reliably with high probability. Other researchers have improved upon von Neumann's techniques to calculate the necessary and sufficient amount of redundancy to perform Boolean functions [Dobrushin and Ortyukov 1977a, 1977b]. These results were improved upon by Pippenger, who showed how Boolean functions may be computed reliably (with constant multiplicative redundancy) by gates susceptible to noise [Pippenger et al. 1991; Pippenger 1989, 1985]. While these earlier papers provided rigorous theoretical frameworks, practical designs which exploit redundancy to achieve reliability while utilizing noise-susceptible CMOS gates have been demonstrated as well.

In the domain of CMOS, the "probability of correctness" of a CMOS device originates from the probabilistic nature of charge transport. Several authors demonstrate methods for improving the noise immunity of logic circuits by adopting design styles based on Markov random fields [Bahar et al. 2003; Nepal et al. 2005]. This research seeks to realize reliable deterministic computation using noise-susceptible CMOS devices. Energy efficiency, performance, and implementing probabilistic applications are not the main considerations of this work.

Using defect-prone components to build reliable computing systems is a well-researched area. Conventional approaches to fault tolerance have included designing redundant systems with reliable arbitrators [Siewiorek and Swarz 1998]. Technology scaling and nanotechnology, with their associated reliability problems, have accelerated research into chip-level fault tolerance and fault avoidance. Fault-tolerance approaches include techniques like speculative execution on faster (but less reliable) logic elements and verification by slower (and more reliable) logic elements [Jacome et al. 2004]. In contrast to all the aforementioned work, our approach does not require redundancy. In fact, "defective" switches are utilized for performing meaningful computation in their normal defective modes of operation.

Thus, we differentiate our work with the theoretical and engineering results summarized earlier as follows: (i) Our work demonstrates the value of harnessing the statistical behavior of gates rather than *overcoming* this behavior, while realizing efficient computation; (ii) while achieving useful computation through noise-susceptible CMOS devices, considerations of energy efficiency and performance are the driving and distinguishing themes of our work; and (iii) to the best of our knowledge, we are the first to present a concrete architecture, a codesign methodology, metrics, and experimental results which demonstrate that gains in the context of real-world applications may be implemented using CMOS devices with probabilistic behavior. We have already demonstrated that PCMOS devices afford energy savings at the device level by tradingoff probability of correctness for energy efficiency [Cheemalavagu et al. 2005, 2004; Chakrapani et al. 2006]. We now show application-level energy and performance benefits of PSOCs using PCMOS technology, as well as algorithm-architecture-technology navigation for efficient PSOC designs.

## 3. PROBABILISTIC SYSTEM-ON-A-CHIP ARCHITECTURES

The central idea behind probabilistic system-on-a-chip (PSOC) architectures is to harness the probabilistic behavior of PCMOS devices to design architectural primitives with well-defined statistical behaviors. These primitives, in turn, implement key (probabilistic) steps of probabilistic algorithms. Probabilistic algorithms, by definition, are those which "toss coins," or execute steps whose outcomes have probabilities associated with them. Examples of such algorithms include the celebrated test for *primality* [Rabin 1976; Solovay and Strassen 1977], used as a key building block in RSA public-key cryptosystems. As we demonstrate in this article, PSOC implementations yield impressive energy and performance benefits at the application level. These energy and performance benefits arise from two sources: (i) the low-voltage (hence low-energy) characteristics of PCMOS technology; and (ii) harnessing the inherent statistical behavior of PCMOS devices directly to perform useful computation, rather than overcoming this statistical behavior to achieve determinism—conventional approaches toward this end are rooted in redundancy or high-voltage operation and inevitably lead to energy and (possibly) performance penalties. In this section, for completeness, we first present a brief overview of PCMOS technology. Following this, we describe PSOC architectures and discuss their performance and energy modeling.

### 3.1 PCMOS Technology

PCMOS devices are CMOS devices whose behavior is probabilistic. Of the several possible techniques for realizing PCMOS devices (some of which are described in Palem et al. [2005]), it has been demonstrated that ambient thermal noise can be used to randomize the behavior of a conventional CMOS device [Korkmaz et al. 2006]. Based on analytical models and HSpice simulation of such devices, it has been shown for a probabilistic inverter that: (i) the switching energy $E$ of a CMOS device grows with probability of correctness $p$ in the interval $1/2 < p \leq 1$ and furthermore, the rate of growth dominates an exponential [Korkmaz et al. 2006]. This behavior is not "incidental," but in fact based on well-studied foundational principles [Palem 2005]. In addition, (ii) for a given probability $p$ of correctness, the noise magnitude (quantified as its RMS value) and corresponding switching energy $E$ of a CMOS device are quadratically related. These two relationships characterize the behavior of PCMOS inverters. While the former serves as a basis for obtaining energy savings by relaxing the correctness constraints on switching, the latter serves as a basis for characterizing and controlling the desired statistical behavior of PCMOS devices. We use these PCMOS switches in architectural building blocks to design PSOC architectures which, in turn, implement applications based on probabilistic algorithms.

### 3.2 PSOC Architectures

As illustrated in Figure 1, PSOC architectures are envisioned to consist of two parts: a *host* processor which consists of a conventional low-energy embedded processor, such as the StrongARM SA-1100 [Corp. 1998], coupled to a coprocessor which utilizes PCMOS technology. As we shall see in this article, such a

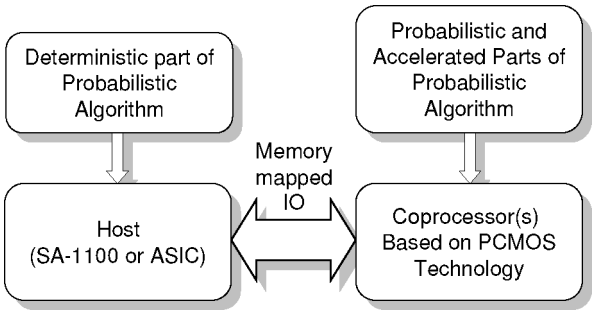


Fig. 1. The canonical PSOC architecture.

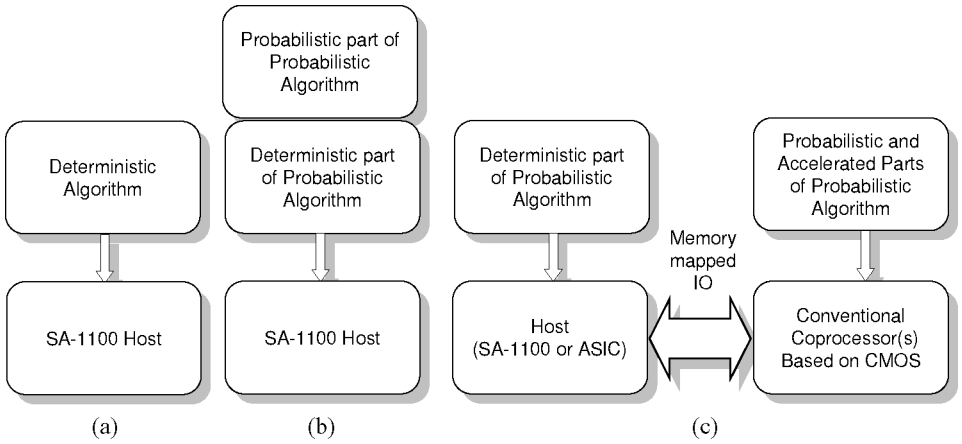


Fig. 2. Conventional implementation alternatives for an application.

host-coprocessor architecture affords several benefits. For a comparative study of the benefits of PSOC architectures with current designs, we consider three choices to be competitors for a PSOC.

—As shown in Figure 2(a), a conventional host-only architecture executes a deterministic algorithm where a deterministic counterpart of the probabilistic algorithm executes completely on the host processor.

—A conventional host-only architecture executes a probabilistic algorithm where the probabilistic algorithm of interest executes completely on the host processor. The probabilistic component utilizes well-known pseudorandom number generators implemented in software [Park and Miller 1988]. This style of implementation is shown in Figure 2(b).

—As shown in Figure 2(c), a conventional SOC has a CMOS-based coprocessor implementing the probabilistic parts of the application, whereas the deterministic parts are executed as software on the host processor.

These cases encompass alternate implementations of the application. Throughout this study, the coprocessors illustrated in Figures 1 and 2(c) are realizations using PCMOS and CMOS, respectively, that are application specific.

### 3.3 Performance and Energy Modeling of PSOC Architectures

Energy consumed (in joules) and performance (in terms of running time in seconds) as the application executes on a particular architecture will be the chief attributes of interest. Our energy and performance modeling is simulation-based. However, the energy consumed by PCMOS devices are derived from actual measurements from a PCMOS test chip. As shown in Figure 1, in a PSOC architecture, the coprocessors are memory mapped and the communication is modeled through LOAD and STORE instructions executed by the host. A special instruction triggers the execution of the application-specific PCMOS coprocessor.

To model the performance of an application executing on such a PSOC, we have modified the Trimaran (http://www.trimaran.org) [Chakrapani et al. 2005] compiler and simulator to reflect the ISA of the StrongARM SA-1100 processor. The simulator records the trace of activity in the SA-1100 host processor, as well as accesses to the coprocessors. This simulation is combined with the performance models of the coprocessor, typically obtained through HSpice simulations, to yield the performance of the application in terms of execution time.

The energy consumed by an application executing on such a PSOC is the sum of energy consumed by the host, the energy consumed by the coprocessor, and energy consumed due to communication between these components. To measure the energy of an application executing on such an architecture, we have incorporated the analytical model of Jouletrack [Sinha and Chandrakasan 2001] into the Trimaran simulator. This model is reported by its authors to be within 3% of the energy consumed by the actual SA-1100 processor. Thus, apart from estimating the performance of an application, the simulator is also used to estimate the energy consumed by the StrongARM host. The latencies caused by the slower PCMOS coprocessor are accounted for as well. To estimate the energy consumed by coprocessors, the latter were designed and synthesized using the associated energy consumption estimated using HSpice. In addition, the actual measurement data of fabricated devices also using TSMC 0.25 $\mu$m technology and their results are used as well. This, combined with the activity trace of the coprocessor (recorded by the simulator) yields the energy consumed in the coprocessor. Our performance and energy modeling techniques for a PSOC are illustrated in Figure 3. Since the applications of interest are probabilistic, at least 50 distinct executions are used to calculate the energy and performance of an application of one of the various alternate realizations (listed in Section 3.2).

### 3.4 Energy and Performance Metrics for PCMOS

To highlight and analyze the benefits of PCMOS technology, we now introduce several metrics to study the gains possible from PSOC implementations. In particular, we will consider the *energy performance product*, or EPP for short, as the chief metric of interest. The EPP metric has been chosen due to several considerations. It captures the chief characteristics of interest, namely, the energy as well as the time needed for the execution of an application. In addition, given an architectural design to implement an application, the application execution could potentially be accelerated by replicating architectural blocks to exploit parallelism. In addition, techniques like voltage scaling could be used to tradeoff

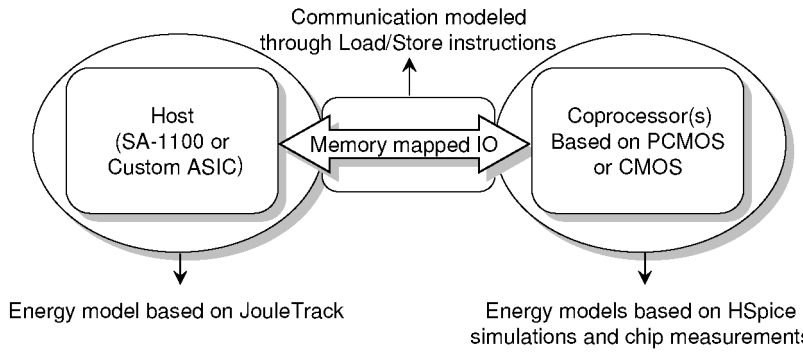


Fig. 3. Energy and performance modeling methodology for each component of a PSOC.

performance for energy efficiency. It is our intention that the EPP metric would remain invariant under replication as well as voltage scaling, as improvements in time would be offset by the increase in energy and vice versa. Hence, EPP is a valuable metric to compare architectural implementations across differing technologies. Given the EPP of two alternate realizations, they can be compared by computing the energy performance product gain.

*Energy performance product gain*: $\Gamma_{\mathcal{I}}$ is the ratio of the EPP of the baseline, denoted by $\beta$, to the EPP of a particular architectural implementation $\mathcal{I}$. Specifically, $\Gamma_{\mathcal{I}}$ is calculated as follows:

$$\Gamma_{\mathcal{I}} = \frac{Energy_{\beta} \times Time_{\beta}}{Energy_{\mathcal{I}} \times Time_{\mathcal{I}}} \tag{1}$$

Initially, to highlight the benefits of PSOC over the case where there is no coprocessor, the baseline will correspond to the case where the entire computation is executed on the SA-1100 host. For example, in the case of a randomized neural network application which solves the vertex cover problem, the baseline will be the case where the SA-1100 host computes both the probabilistic and deterministic parts of the application (as illustrated in Figure 2(b)) and $\mathcal{I}$ corresponds to the case where the core probabilistic step is computed using a PCMOS-based coprocessor and the rest of the computation is performed using an SA-1100 host (as illustrated in Figure 1). Later, to quantify the benefits of PSOC implementations over conventional CMOS-based SOC implementations, the baseline will correspond to the case where the SA-1100 host is coupled to a functionally identical CMOS-based coprocessor (as in Figure (c)), where the coprocessor computes the core probabilistic step. Wherever we present EPP gain results, we will explicitly mention the baseline.

## 4. THE A$^2$T CODESIGN FRAMEWORK

As mentioned in the Introduction, the gains obtained by leveraging PCMOS technology are due to a unique codesign methodology that exploits the technology characteristics of PCMOS, as well as algorithm characteristics of the application drivers, to provide a "good-fit" implementation in the form of a PSOC architecture. Since the codesign methodology is of greater interest than a detailed description

Table I. Algorithms, Applications Based on Them, and Core Probabilistic Step for Each Algorithm

| Algorithm | Application Scenarios | Implemented(s) Application(s) | Core Probabilistic Step |
|---|---|---|---|
| Bayesian Inference [MacKay 1992] | SPAM Filters, Cognitive applications, Battlefield Planning [Pfeffer 2000] | Windows printer trouble shooting, Hospital Patient Management [Beinlich et al. 1989] | Choose a value for a variable from a set of values based on its conditional probability |
| Random Neural Network [Gelenbe 1989] | Image and pattern classification, Optimization of NP-hard problems | Vertex cover of a graph | Neuron firing modeled as a Poisson process |
| Probabilistic Cellular Automata [Wolfram 2002] | Pattern classification | String classification [Fuks 2002] | Evaluating the probabilistic transition rule |
| Hyper-Encryption [Ding and Rabin 2002] | Security | Message encryption | Generation of a random string and encryption pad generation from this string |

of the application drivers and their implementation details, the rest of this section is organized as follows: We briefly introduce the applications of interest and their characteristics that play an important role in codesign (a detailed description of each of the algorithms, the specific partitioning strategy for each of these applications, and the corresponding PSOC implementation details are presented in Section 5). We then present the energy and performance results obtained from the PSOC implementation and a comparative study using the metrics introduced in Section 3.4. We analyze these gains and then describe the algorithm and technology characteristics that influence the codesign, and future directions for obtaining better gains.

### 4.1 A Brief Description of the Applications of Interest

We consider applications based on probabilistic algorithms drawn from the cognitive and security domains. The algorithms include *Bayesian inference* [MacKay 1992], *probabilistic cellular automata* [Fuks 2002], *random neural networks* [Gelenbe 1989], and *hyperencryption* [Ding and Rabin 2002]. These algorithms will be referred to as BN, PCA, RNN, and HE, respectively. The applications in which each is utilized are summarized in Table I.

Table II. Maximum and Minimum EPP Gains of PCMOS over Baseline Implementation

| Algorithm | $\Gamma_{\mathcal{I}}$ | |
|---|---|---|
| | Min | Max |
| BN | 3 | 7.43 |
| RNN | 226.5 | 300 |
| PCA | 61 | 82 |
| HE | 1.12 | 1.12 |

These ERP gains are where the implementation $\mathcal{I}$ has a StrongARM SA-1100 host and a PCMOS-based coprocessor.

The PSOC implementation for each of the algorithms consists of a StrongARM SA-1100 host and an application-specific coprocessor, as mentioned in Section 3.2. The coprocessor design involves the partitioning of each of these applications between the host and application-specific PCMOS-based coprocessor. Once partitioned, PCMOS-based coprocessors are designed by hand. Though the exact manner in which these applications are partitioned varies and is not (currently) automated, all variations thereof follow a common theme. Common to these applications (and to almost all probabilistic algorithms) is the notion of a core probabilistic step with its associated probability parameter $p$. For example, in one probabilistic cellular automata application [Fuks 2002], this step is the probabilistic transition of an automaton which decides its next state based on the current state and a probability parameter $p$ associated with the transition rule. The core probabilistic step for each application of interest is presented in Table I. For each of the candidate applications, this core probabilistic step is identified both by hand and by the PCMOS-based coprocessors designed for it. The deterministic parts of the application (e.g., choosing which transition rule to apply in the context of probabilistic cellular automata) are implemented as software executing on the host processor.

### 4.2 Application-Level Gains of PSOC

Table II summarizes the application-level EPP gains of PSOC over the baseline for each application of interest. Gains at the scope of an entire application range from a factor of about 80 for the PCA application to a factor of about 300 in the context of the RNN application. As mentioned earlier, the baseline implementation for BN, HE, PCA, and RNN applications is the StrongARM SA-1100 computing the deterministic as well as the probabilistic content, and $\mathcal{I}$ is a PSOC executing an identical probabilistic algorithm.

As can be seen from Table II, the application-level gains of each application vary. For example, in the RNN case, a range of EPP gains are observed whenever multiple data points are available. This is attributable to the probabilistic nature of the applications: Their execution characteristics differ, yielding disparate gains for various input sets and sizes. Next, we analyze the factors affecting gains in a systematic way.

### 4.3 An Analysis of Gains Due to PCMOS

Intuitively, application-level gains in energy and performance depend on two factors: (i) the "amount of opportunity" in the application to leverage the

Table III. Application Level Flux, Maximum, and Minimum, EPP Gains of PCMOS Over the Baseline Implementation

| Algorithm | Flux $\mathcal{F}$ (as percentage of total operations) | $\Gamma_{\mathcal{I}}$ | |
|---|---|---|---|
| | | Min | Max |
| BN | 0.25%–0.75% | 3 | 7.43 |
| RNN | 1.64%–1.97% | 226.5 | 300 |
| PCA | 4.19%–5.29% | 61 | 82 |
| HE | 12.5% | 1.12 | 1.12 |

These are measurements where the implementation $\mathcal{I}$ has a StrongARM SA-1100 host and a PCMOS-based coprocessor.

PCMOS-based coprocessor; and (ii) the amount of gain afforded "per unit of opportunity." Broadly, the factors which influence gain can be classified as either *implementation-independent* (which include algorithmic, amount of opportunity, e.g., the inherent in an algorithm) and *implementation-dependent* characteristics (including technology and architecture characteristics which influence the amount of gain afforded per unit of opportunity). These algorithmic, architecture, and technology characteristics in turn influence the codesign methodology (hence the name $A^2T$ codesign methodology); these considerations are outlined to follow and the specific effect on PSOC design for each application of interest will be described in Section 5.

4.3.1 *Implementation-Independent Characteristics Influencing* PSOC *Design and Gains.* As mentioned before, the core probabilistic step of each application is implemented in the PCMOS-based coprocessor, and one core probabilistic step will be regarded as one "unit of opportunity." The core probabilistic step for each application has been presented in Table I. Given this, it is natural to expect that the higher the opportunity to exploit PCMOS technology for efficient implementations, the higher the gains. The amount of opportunity is formalized through the notion of *probabilistic flux* $\mathcal{F}$ (or flux for short), where $\mathcal{F}$ of an algorithm is defined as the ratio of the core probabilistic steps to the total number of operations of an algorithm during a typical execution. The "total number of operations" in this context refers to the total number of cycles consumed by the deterministic instructions executing on the StrongARM processor. Informally, $\mathcal{F}$ can be regarded as the ratio of the number of times a PSOC coprocessor will be invoked to the number of times the host processor is "invoked" (i.e., cycles executed by the host processor). Flux for various algorithms will be presented in either ratio form or in the form of a percentage.

With this as background and revisiting Table II, we observe that the application-level gains application vary. For example, in the BN case, a range of EPP gains is observed whenever multiple data points are available. Table III presents the flux as well as the min and max gains for each of the applications.

The variation in gain is attributable to the probabilistic nature of the applications under consideration. Since these applications are probabilistic, their execution characteristics (and hence the flux) depend on the input size and the actual inputs. To understand the effect of flux, let us consider the BN application

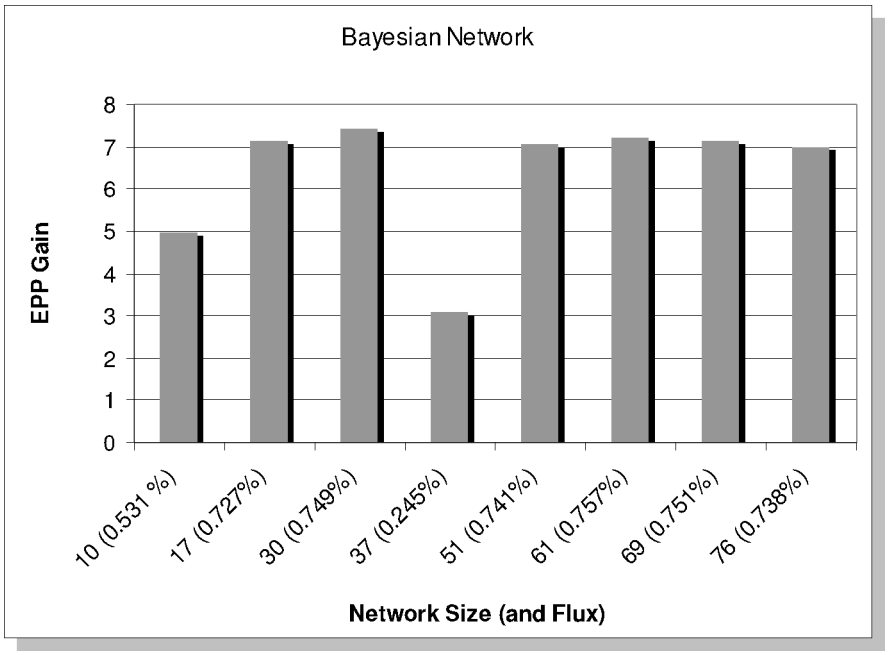


Fig. 4. Gain and flux for Bayesian networks of various sizes.

in detail. Figure 4 shows the gain for each network size and the corresponding flux $\mathcal{F}$.

As expected, as the flux increases from 0.25 % (for a Bayesian network size of 37) to 0.75 % (for a Bayesian network size of 69), the corresponding gain increases from a factor of 3 to a factor of 7.14. In general, for a specific application, consider the energy consumed by the baseline implementation. This is a sum of the energy consumed at the StrongARM host for executing the deterministic parts of the application $Energy_{det,\beta}$ and that consumed at the StrongARM host for executing the probabilistic part of the application $Energy_{prob,\beta}$.

$$Energy_{\beta} = Energy_{det,\beta} + Energy_{prob,\beta}$$

Consider $Energy_{det,\beta}$ to be the energy consumed by the baseline for executing the deterministic part of the application. If the average energy consumed per invocation of the host processor (i.e., per cycle of the host processor) is $Energy_{cycle,host}$ and the number of invocations of the host processor is $Cycles_{det,host}$, then

$$\begin{aligned} Energy_{\beta} &= Energy_{det,\beta} + Energy_{prob,\beta} \\ &= Cycles_{det,host} \times Energy_{cycle,host} + Energy_{prob,\beta}. \end{aligned}$$

Consider $Energy_{prob,\beta}$ to be the energy consumed by the baseline for executing the probabilistic part of the application. Let the energy consumed per invocation of the core probabilistic step be $Energy_{flux,\beta}$. From the definition of flux $\mathcal{F}$, the

Table IV. EPP Gain of PCMOS Over SA-1100 and Over CMOS for the Core Probabilistic Step

| Application | Gain Over SA-1100 | Gain Over CMOS |
|---|---|---|
| BN | $9.99 \times 10^7$ | $2.71 \times 10^6$ |
| RNN | $1.25 \times 10^6$ | $2.32 \times 10^4$ |
| PCA | $4.17 \times 10^4$ | $7.7 \times 10^2$ |
| HE | $1.56 \times 10^5$ | $2.03 \times 10^3$ |

number of invocations of the core probabilistic step is $\mathcal{F} \times Cycles_{det,host}$ and

$$\begin{aligned} Energy_\beta &= Energy_{det,\beta} + Energy_{prob,\beta} \\ &= Cycles_{det,host} \times Energy_{cycle,host} + Energy_{prob,\beta} \\ &= Cycles_{det,host} \times Energy_{cycle,host} + \mathcal{F} \times Cycles_{det,host} \times Energy_{flux0,\beta}. \end{aligned}$$

Similarly, the energy consumed by the PSOC implementation $Energy_\mathcal{I}$ can be written as

$$\begin{aligned} Energy_\mathcal{I} &= Cycles_{det,host} \times Energy_{cycle,host} + \mathcal{F} \times Cycles_{det,host} \times Energy_{flux,\mathcal{I}} \\ &\approx Cycles_{det,host} \times Energy_{cycle,host}. \end{aligned}$$

The approximation arises due to the fact that the PCMOS-based coprocessor consumes almost negligible energy (this can be seen from Table IV; however, the actual gains presented here consider the energy of the coprocessor as well, and the approximation is used purely for explanation purposes). Similarly, we can derive an expression for performance, and for a specific application the gain $\Gamma_\mathcal{I}$ can be characterized as

$$\begin{aligned} \Gamma_\mathcal{I} &= \frac{Energy_\beta \times Time_\beta}{Energy_\mathcal{I} \times Time_\mathcal{I}} \\ &= \left(1 + \frac{\mathcal{F} \times Energy_{flux,\beta}}{Energy_{cycle,host}}\right) \times \left(1 + \frac{\mathcal{F} \times Time_{flux,\beta}}{Time_{cycle,host}}\right). \end{aligned} \tag{2}$$

Harking back to Section 4.3, we notice that the gains depend on flux $\mathcal{F}$, an implementation-independent algorithmic characteristic which determines the "amount of opportunity." Also, the gains depend on $Energy_{flux,\beta}$ and $Time_{flux,\beta}$, which are implementation, dependent technology and architecture characteristics. Counterintuitively, the gains also depend on $Energy_{cycle,host}$ and $Time_{cycle,host}$, which capture the efficiency *of the host processor*! This will be further explored in Section 4.4.

For the Bayesian network application, Figure 5 shows how $\Gamma_\mathcal{I}$ varies with the flux. The line is analytically calculated using Eq. (2), and the points correspond to actual values measured using the simulations. Two particular points of interest, whose flux correspond to Bayesian network sizes of 37 and 69 nodes, respectively, are also shown in the figure. It can be seen that the simulation result matches closely with that of the analytical model. Similarly, Figure 6 shows the variation of $\Gamma_\mathcal{I}$ with the flux for a randomized neural network application. Again, the line is calculated analytically and the points correspond to gains obtained from simulation. Thus, the flux $\mathcal{F}$ of an algorithm is an important

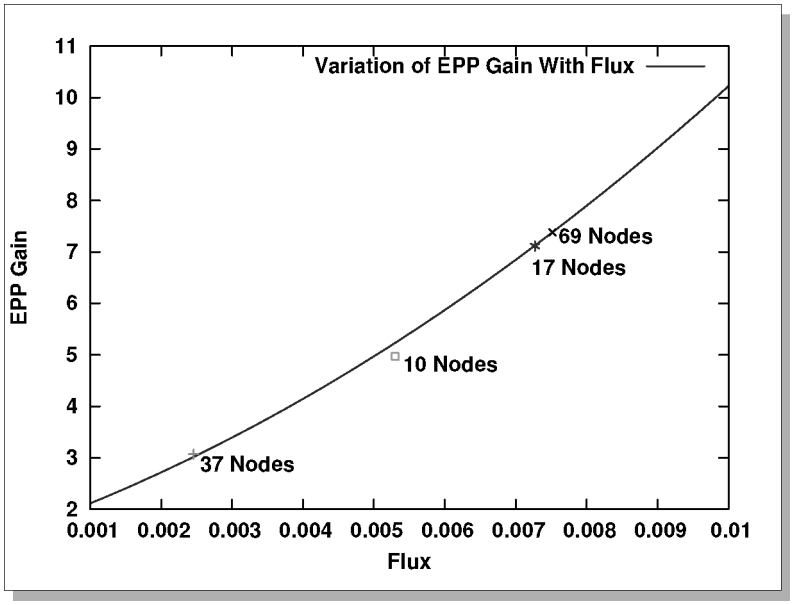


Fig. 5. Variation of gain with respect to flux for a Bayesian network.

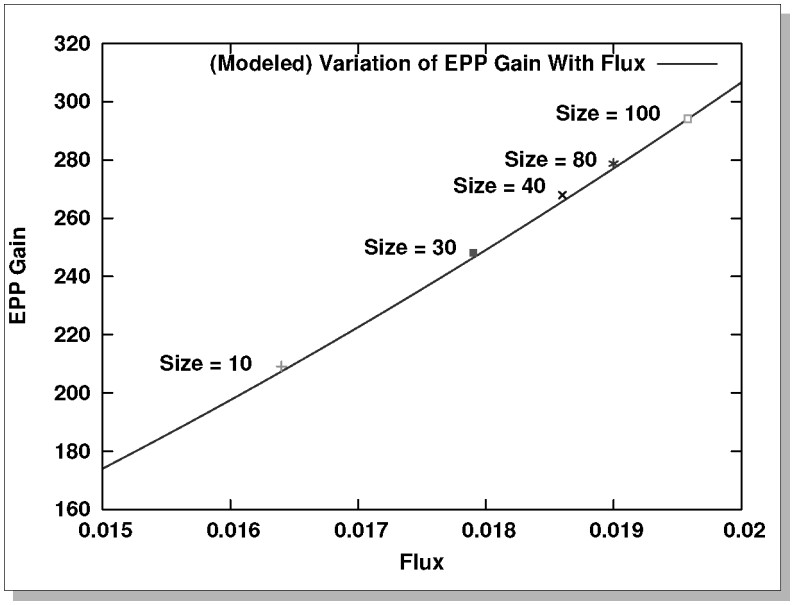


Fig. 6. Variation of gain with respect to flux for a randomized neural network.

characteristic that determines the gains derived from a PSOC implementation. Hence, given an algorithm, it is advantageous to maximize opportunity (in this context, increase the amount of probabilistic steps whenever possible) and given an application, to leverage higher gains, it is advantageous to leverage an

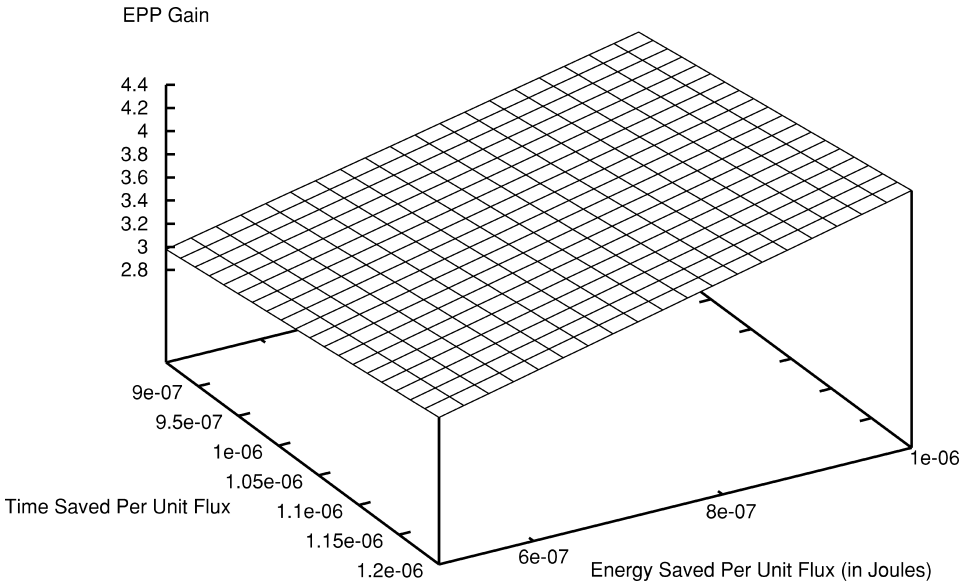


Fig. 7. For a fixed flux, variation of gain with respect to energy saved per unit flux and time saved per unit flux by using PCMOS.

algorithm with the highest "probabilistic content," or flux. These considerations influence the selection and optimization of the algorithm used for a particular application in our $A^2T$ codesign methodology.

4.3.2 *Implementation-Dependent Characteristics Influencing* PSOC *Design and Gains.* Application-level gains not only depend on the flux of an application, but on the energy and performance gains afforded per unit of opportunity. Table IV presents the EPP gain of a PCMOS-based coprocessor for the core probabilistic step of each of the applications of interest. The second column in the table corresponds to the case where $\beta$ is the SA-1100 host without any coprocessor and the third column to the case where $\beta$ is a SA-1100 host coupled to a conventional CMOS-based coprocessor. As can be seen from the table, a PCMOS-based coprocessor is over five orders of magnitude better in terms of EPP when compared to an SA-1100 processor, and over three orders of magnitude when compared to a CMOS-based coprocessor while executing the core probabilistic step of the HE application.

For a given flux, the application-level gain will increase with an increase in the energy, as well as performance gain, per unit flux. To illustrate, let us revisit the Bayesian network application and the gain $\Gamma_{\mathcal{I}}$, where $\mathcal{I}$ is a PSOC and the baseline is a StrongARM SA-1100 host without a coprocessor. In particular, let us consider the case where the size of Bayesian network is 37 nodes and the corresponding flux is 0.25%. Now, the higher the efficiency of PCMOS (in the form of lesser energy and faster execution) per invocation of the core probabilistic step, the higher the gain. In other words, the more energy and time saved per invocation of the core probabilistic step, the higher the gain afforded by the PSOC implementation. Figure 7 illustrates a variation of $\Gamma_{\mathcal{I}}$ with respect

to the cycles per unit flux and energy per unit flux expended by the baseline implementation. The point where the surface intersects the $z$ axis presents the performance and energy consumption per unit flux, which corresponds to a gain of 3, and the point plots the performance and energy consumption per unit flux for a Bayesian network of size 37.

With this as background and revisiting Table IV, we observe that the energy and performance gain afforded per unit of flux vary across applications. This is an artifact of both the functionality of the core probabilistic step as well as the characteristics of PCMOS technology. The characteristics of PCMOS technology which influence the energy and performance gains per core probabilistic step are enumerated in the following:

—PCMOS *energy-efficiency*. PCMOS-based switches are extremely efficient for implementing logical operations with probabilities associated with their outcomes. For example, the energy consumed for one *probabilistic inversion* (i.e., a logical NOT operation with an associated probability of correctness $p$) operation is 0.4 pico-joules [Cheemalavagu et al. 2005], whereas emulating similar functionality using a hardware-based implementation of the Park-Miller algorithm consumes 2,025 times this much energy. As expected, more complex core probabilistic operations afford high gains per unit flux.

—PCMOS *specialization*. Apart from efficient operation, PCMOS devices can be "tuned" to the desired probability parameter of any probabilistic step $S$. For example, PCMOS-based primitives can be built for probabilistic inversion with a probability of correctness $p = 0.75$. Further details as to how the probability of correctness can be controlled are presented in Cheemalavagu et al. [2005].

Corresponding implementations in software or a conventional CMOS incur a penalty for nontrivial probabilities ($p \neq 0.5$). This is because to achieve, say, a probability parameter $p = 0.75$, typical implementations would generate a number uniformly at random, say between 0 and $2^{16}$, and compare it with $2^{16} \times 0.75$. This involves *dilation* of 1 bit to 16 bits captured by the notion of the *dilation factor* $\mathcal{D}$. Hence, core probabilistic steps with nontrivial probabilities afford higher gains per unit flux.

—PCMOS *replication due to specialization.* Whereas specialization to a particular probability parameter $p$ has the advantage of avoiding the penalties associated with tuning and dilation, separate PCMOS building blocks need to be implemented for probabilistic operations that are similar, and differ only in their probability parameter. For example, two different PCMOS-based primitives need to be built for two probabilistic inversion operations with probability $p = 0.75$ and $p = 0.80$, respectively. This replication of PCMOS primitives due to specialization is captured by the metric *spread factor*, denoted by $\mathcal{S}$, and is a count of such distinct probability parameters used by an application. Spread factor guides application optimization by reducing the distinct probability parameters used by an application, and architecture optimization by choosing a nonspecialized implementation if the spread factor is too high.

—PCMOS *operating frequency*. Though PCMOS devices are extremely (energy) efficient, the operating frequencies of our current implementations are low [Cheemalavagu et al. 2005], at about 1 MHz. This characteristic acts as

a potential limitation to the peak rate (with which probabilistic steps can be executed on the PCMOS-based coprocessor. Given this limitation, the peak rate (with which a probabilistic step $S$ needs to execute on the coprocessor so that the host processor is not stalled) is a characteristic of interest. This peak rate will henceforth be referred to as the *application demand rate for the probabilistic step* $S$. Intuitively, the application demand rate is dependent on algorithm characteristics and the operating frequency of the host processor. If the application demand rate for a probabilistic step $S$ is higher than the operating frequency of the PCMOS building block which executes the step $S$, the host processor will need to *stall* until the probabilistic steps finish execution. This is analogous to memory stall cycles in modern microprocessors where there is a mismatch between the frequency of operation of the data path and the memory subsystem. This limitation can be remedied through parallelism, that is, by replicating the PCMOS building block which executes the step $S$. The number of replications is captured through the *replication factor* $\mathcal{R}$. The replication factor is a characteristic that guides both application and architecture optimization. On the application side, program transformations can be performed to better interleave probabilistic with deterministic steps (which execute on the host processor) so that the peak application demand rate is reduced. In addition, since the current implementation of PCMOS devices does not allow them to be switched off when not needed (akin to clock gating techniques in conventional microarchitecture design), increased replication, while decreasing the time consumed to execute an application, might increase the energy consumption. This tradeoff needs to be taken into account while replicating PCMOS building blocks.

—PSOC *communication costs*. There is an inherent cost of communication between the host processor and PCMOS-based coprocessor, which can potentially reduce the gains. When partitioning the application, this should be considered as well.

### 4.4 A Comparison of PSOC with Conventional SOC Designs

So far, we have demonstrated the utility of PCMOS technology and PSOC implementations of selected applications by presenting the energy and performance gains of PCMOS-based PSOC designs over a conventional host-only style of implementation. A more ambitious and interesting comparison would be with that of a conventional SOC design where a functionally identical coprocessor is designed with conventional CMOS technology. With a conventional CMOS-based SOC as the baseline, the gain $\Gamma_{\mathcal{I}}$, where $\mathcal{I}$ is a PCMOS-based PSOC for a HE as well as a PCA application, is 1.

This is in spite of high flux and gains per core probabilistic step in the corresponding applications. To explain this, let us revisit Eq. (2). We note that the gains depend on the flux $\mathcal{F}$, namely, the gains per core probabilistic step (approximately $Energy_{flux,\beta}$ and $Time_{flux,\beta}$) which were studied and analyzed in the preceding sections. More importantly, the gains depend on $Energy_{cycle,host}$ and $Time_{cycle,host}$ as well, which indicates that if the computation that is executed on the SA-1100 host dominates the energy and time consumption of the

entire application, *then the gains from a* PCMOS*-based* PSOC *will be low*. Hence, even though the proportion of core probabilistic steps in the entire application is high, as well as the gains per core probabilistic step, using a PCMOS-based coprocessor has almost no impact in trems of application-level time and energy consumption. Thus, the gains through PCMOS—the limits being substantial, as shown in Table IV—can be truly achieved only if the amount of effort spent in the coprocessor is comparable, in terms of EPP units, to that spent in the host.

To verify this hypothesis, a baseline SOC architecture in which the host processor and coprocessor are both custom ASIC architectures is considered. With this notion, moving away from a StrongARM host processor to one realized from custom ASIC logic, the amount of energy and running time spent in the host are considerably lower. Thus, and perhaps counterintuitively, increasing the efficiency of the competing approach enhances the value of PCMOS gains at the application level. In the context of the HE application, and with this change to the baseline, the gain $\Gamma_{\mathcal{I}}$ increases to 9.38; almost an order of magnitude. Similarly, when a baseline with a custom ASIC host is used, the $\Gamma_{\mathcal{I}}$ value in the context of the probabilistic cellular automata application increases to 561. In all of these comparisons, the CMOS-based coprocessor operated at an optimal frequency, that is, the frequency which yields the lowest energy consumption without degrading application performance. In addition, CMOS-based coprocessors are assumed to leverage techniques like clock gating with no overhead. In this respect the gain estimates are conservative. We view this fact as being extremely favorable for PSOC-based designs, since as host processors become more efficient in future technology generations, the gains of PSOC over conventional SOC architectures increase.

## 5. THE SUITE OF APPLICATIONS, PARTITIONING, OPTIMIZATION, AND PSOC IMPLEMENTATION

In this section, we describe in detail the applications, their partitioning, optimization, and PSOC implementation.

—*Bayesian networks (BN)*. Bayesian inference [MacKay 1992] is a statistical inference technique. Hypotheses, their corresponding probability weights; and evidences are central components of this technique. The probability weight $p$ associated with a hypothesis $H$ is interpreted as the *degree of belief* in the hypothesis. Evidences support (or discount) a hypothesis, thereby increasing (or decreasing) the associated probability weight, and hence the degree of belief in the hypothesis. Hypotheses whose probability weights approach 1 are most likely and those whose probability weights approach 0 are very unlikely. A Bayesian network can be used to perform Bayesian inference. A Bayesian network is a directed acyclic graph $G$ of nodes $V$ which represent variables and edges $E \subseteq V \times V$, which in turn represent dependence relations between the variables. Each node $v_x \in V$ can be associated with a value from a finite set of values $\Sigma_x$. The set $\Sigma_x$ will be referred to as the *set of possible values associated with* $v_x$.

Without loss of generality, let $v_1, v_2, v_3, \ldots, v_k$ be the $k$ parents of $v_x$. Let $\Sigma_1$ be the set of possible values associated with $v_1$; similarly, let $\Sigma_2, \Sigma_3, \ldots, \Sigma_k$ be

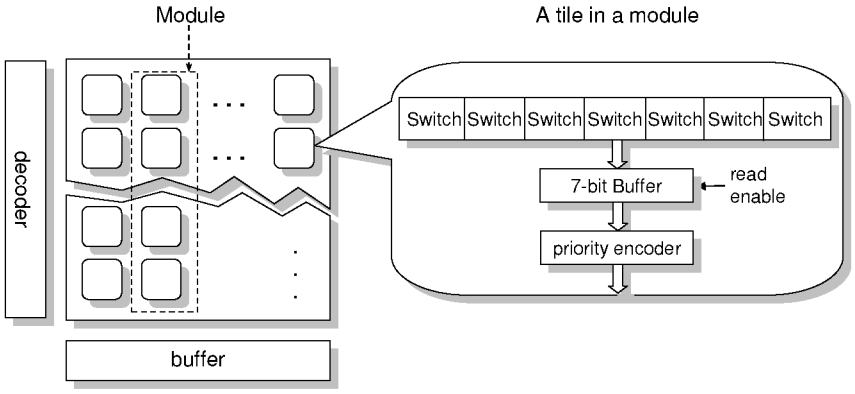


Fig. 8. The coprocessor architecture for a PSOC implementing Bayesian inference.

associated with $v_2, v_3, \ldots, v_k$, respectively. Each value $\sigma \in \Sigma_x$ has a conditional probability $p(\sigma/\sigma' \in \Sigma'_x)$ associated with it, where $\Sigma'_x = \Sigma_1 \times \Sigma_2 \times \Sigma_3 \cdots \Sigma_k$. In essence, $\sigma'$ is the string of values of the variables represented by the $k$ parents of the node $v_x$, and $\Sigma'_x$ is the set of all possible strings. Variables whose values are known a priori are called evidences, and based on evidences, other variables can be *inferred*. Based on network topology and conditional probabilities associated with the variables, various cognitive tasks can be performed. The particular Bayesian networks considered in this study are a part of the following applications: a hospital patient monitoring system [Beinlich et al. 1989] and a printer trouble shooting application for the Windows operating system.

—*Partitioning, optimization, and* PSOC *implementation*. We choose the likelihood weighting algorithm [Pfeffer 2000] for Bayesian inference. To illustrate, consider a node $v_x \in V$ with $\Sigma_x = \{0, 1, 2\}$. As before, let $\Sigma'_x$ be the set of all possible strings of values associated with the parents of $x$. Let $0 \leq p(0/\sigma'), p(1/\sigma'), p(2/\sigma') \leq 1$, where $p(0/\sigma') + p(1/\sigma') + p(2/\sigma') = 1$, be the conditional probabilities associated with $0, 1, 2 \in \Sigma_x$, respectively, given that $\sigma'$ is the string formed by the outputs of parents of the node $v_x$. The inference process performs a random experiment with three possible outcomes 0, 1, or 2 with the associated probability $p(0/\sigma')$, $p(1/\sigma')$, or $p(2/\sigma')$, respectively.

In our PSOC architecture, Bayesian inference will be performed by three PCMOS switches $A$, $B$, and $C$ which correspond to 0, 1, and 2, respectively. The inputs for these switches are fixed at 0 and the probability of correctness associated with $A, B, C$ is by design $p(0/\sigma')$, $\frac{p(1/\sigma')}{1-p(0/\sigma')}$, and 1, respectively. Thus, when the switches are inspected in the order $< A, B, C >$, that value which corresponds to the leftmost switch whose output is the value 1 is the value inferred by the node. In the PSOC design, the set of switches $\{A, B, C\}$ will be referred to as a *row*. A row of switches is associated with each member of the set $\Sigma'_x$, hence at least $|\Sigma'_x|$ rows are required for a node $v_x$. This set of rows associated with a node $v_x$ will be referred to as a *module* which corresponds to the node $v_x$.

As shown in Figure 8, the PCMOS module which corresponds to a node $v_x$ implements a table. Rows in this module are indexed by a particular string $\sigma'$ of

values associated with the parents of $v_x$. The number of columns in the module is $|\Sigma_x|$, where each column corresponds to a value from the set $\Sigma_x$; in our example, $|\Sigma_x| = 3$ (and in the figure, it is 7). A switch in the module, identified by <row, column>, is a specialized PCMOS switch whose probability of correctness is computed as indicated previously. Finally, a conventional priority encoder is connected to the outputs of a row to determine the final result of the random experiment; it performs the function of inspecting the values of a row and choosing the final output associated with $v_x$. The random experiment (used for inference) in this probabilistic algorithm is implemented in the PCMOS coprocessor (which consists of several modules), with the remainder implemented as software executing on the host.

—*Random neural network (*RNN*)*. Following Gelenbe [1989], a random neural network consists of neurons and connections between the neurons. Information is exchanged between neurons in the form of *bipolar signal trains*. Neurons have associated potentials which are defined to be the sums of incoming signals. These potentials, in turn, influence the rate of firing. A random neural network can be modeled as an undirected graph $G$ of nodes (i.e., neurons) $V$ and directed edges (i.e., connections) $E \subseteq V \times V$. Each node has an associated potential $\psi$ which is incremented (decremented) by incoming (outgoing) firings. The firings occur at a constant rate with exponentially distributed intervals. When a node fires, its potential is decremented by one and the polarity and destination of the firing are determined by probability parameters $p_i$ and $p_d$, respectively. Through a suitable combination of network topology, probability parameters, and firing rates, several optimization problems can be solved. The particular neural network considered in this study is used to heuristically determine the vertex cover of a graph due to Gelenbe and Batty [1992].

—*Partitioning, optimization, and* PSOC *implementation*. The Poisson process which models the "firing" of a neuron is implemented in the PCMOS coprocessor with the rest of the computation (distributing the firings, updating the potentials) implemented to execute on the host processor. To realize the Poisson process characterizing a neuron firing, the Bernoulli approximation of a Poisson process [Feller 1984] is used. As an example of a methodological step in our $A^2T$ codesign approach, since the rate at which neuron firings need to be modeled exceeds the rate at which PCMOS-based switches can compute, the PCMOS-based devices which model the Poisson process are replicated to match the required rate. In the interest of efficiency and as another example of our $A^2T$ methodology, the application is restructured to reduce the replication factor $\mathcal{R}$, by interleaving the modeling of neuron firings (in the PCMOS-based coprocessor) and the processing of these firings (in the host processor), thus distributing the firings more evenly over the course of the entire application's execution. This has the effect of reducing the peak application demand bandwidth.

—*Probabilistic cellular automata*. These are a class of cellular automata used to model stochastic processes. Cellular automata consist of cells with local (typically nearest-neighbor) communication. Each cell is associated with a state and a simple transition rule which specifies the next state of a state transition

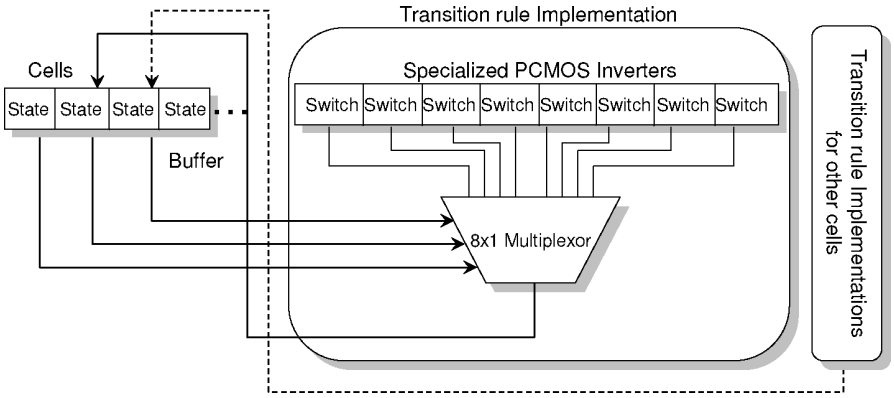


Fig. 9. The custom ASIC host and PCMOS coprocessor architecture for a PSOC implementing a probabilistic cellular automata algorithm.

based on its current state and those of its neighbors. In the probabilistic string classification algorithm [Fuks 2002], the state of each cell is either 0 or 1. The next state of a cell depends on its current state and those of its left and right neighbors. Thus, there are 8 possible transition rules where each rule has two possible outcomes: 0 or 1. In addition, the transition rules are probabilistic: For a transition rule $\tau_i$, there is a $(0 \leq i \leq 7)$ probability that the output state of the rule is 0, denoted by $p_{i,0}$, and the probability that the output state is 1 is denoted by $p_{i,1}$.

—*Partitioning, optimization, and* PSOC *implementation*. Each transition rule is implemented by a PCMOS inverter whose input is 0. The $i$th inverter corresponds to the $i$th transition rule and the probability of correctness associated with the $i$th inverter is $p_{i,1}$. The control-intensive part of choosing the transition rule (based on the state of a cell and those of its neighbors) and updating the states is implemented on the host processor. Since the rate at which the transition rules need to be evaluated exceeds the frequency of operation of PCMOS devices (choosing the transition rule and updating the current state can be executed very quickly on the SA-1100 host), this structure is replicated many times.

In addition, a custom CMOS-based ASIC can be designed to implement the deterministic part of the algorithm. As shown in Figure 9, the ASIC consists of an $n$-bit buffer and $n$ $8 \times 1$ multiplexers, one for each cell. The "select" input of the $j$th multiplexer $0 < j < n + 1$ is the current state of the $j$th cell and the states of its left and right neighbors. The inputs of multiplexers are from PCMOS inverters specialized to the corresponding probability parameters. The transitions are stopped as soon as all the cells have identical states. This is detected by an $n$-input OR and an $n$-input AND gate. The energy of this custom ASIC is obtained through HSpice simulations with the energy of PCMOS inverters obtained from actual chip measurements.

## 6. CONCLUDING REMARKS

So far, we have demonstrated how the statistical behavior of noise-susceptible switches can be harnessed for useful and efficient implementation of a range

of applications. In this section, we explore other implementation and manufacturing concerns and sketch directions for future work.

### 6.1 Other Applications of Interest

Thus far, we have considered instances from the domain of embedded and probabilistic applications. Applications, in general, can be partitioned into three categories: (i) applications which benefit from (or harness) probabilistic behavior at the device level naturally (which is the main category of interest in this article); (ii) applications that can tolerate (and tradeoff) probabilistic behavior at the device level (but do not need such behavior naturally); and (iii) applications which cannot tolerate probabilistic behavior at all. It is interesting to note that PCMOS technology can be effectively utilized for the second category of applications as well. Applications from the domain of digital signal processing provide a natural tradeoff between energy consumed and quality of solution, which can be characterized in the form of a signal-to-noise ratio (SNR). For further details about implementing DSP applications using PCMOS technology, the reader is referred to George et al. [2006].

### 6.2 Towards Design Automation

As mentioned in Section 3, the central idea behind probabilistic system-on-a-chip architectures is to design architectural primitives with statistical behavior. These primitives, in turn, implement key (probabilistic) steps of probabilistic algorithms. The energy and performance gains presented arise from two sources: (i) the low-energy operation of PCMOS-based switches; and (ii) leveraging the statistical behavior of PCMOS switches to implement probabilistic steps in probabilistic algorithms. The latter can be formalized to design more efficient architectural primitives for probabilistic steps of probabilistic algorithms.

Consider any probabilistic step in a probabilistic algorithm. The abstract model of a probabilistic step is a probabilistic truth table, where for each input *every possible* output is realized with an associated probability. Figure 10 illustrates the probabilistic truth table for one such step in Bayesian inference. The column marked $xyz$ depicts the 3-bit input and columns marked $ab$ represent the 2-bit output, with the probability of obtaining that output presented in parenthesis. Intuitively, realizing such probabilistic truth tables using probabilistic switches is inherently more efficient with PCMOS switches than with conventional (deterministic) CMOS switches (where the probabilistic operation has to be "emulated" using pseudorandom number generators). Given such a truth table, a methodology to automate the design of a Boolean circuit can be innovated. A preliminary (by hand) design for a Boolean circuit which realizes such a probabilistic truth table has been presented in Figure 11.

### 6.3 Reducing Multiple Voltage Levels

In the designs described in this work, the probability $p$ of correctness needs to be varied on an application, specific basis. In addition, an application may use several distinct probability parameters. This, as described in Section 4.3.2, increases the spread factor due to replication caused by specialization. In

| Input xyz | Output with corresponding probability parameters ab | | |
|---|---|---|---|
| 000 | 00 (0.98) | 01 (0.01) | 10 (0.01) |
| 001 | 00 (0.01) | 01 (0.98) | 10 (0.01) |
| 010 | 00 (0.01) | 01 (0.01) | 10 (0.98) |
| 011 | 00 (0.98) | 01 (0.01) | 10 (0.01) |
| 100 | 00 (0.98) | 01 (0.01) | 10 (0.01) |
| 101 | 00 (0.69) | 01 (0.30) | 10 (0.01) |

Fig. 10. The probabilistic truth table for a node in a Bayesian network with 37 nodes.

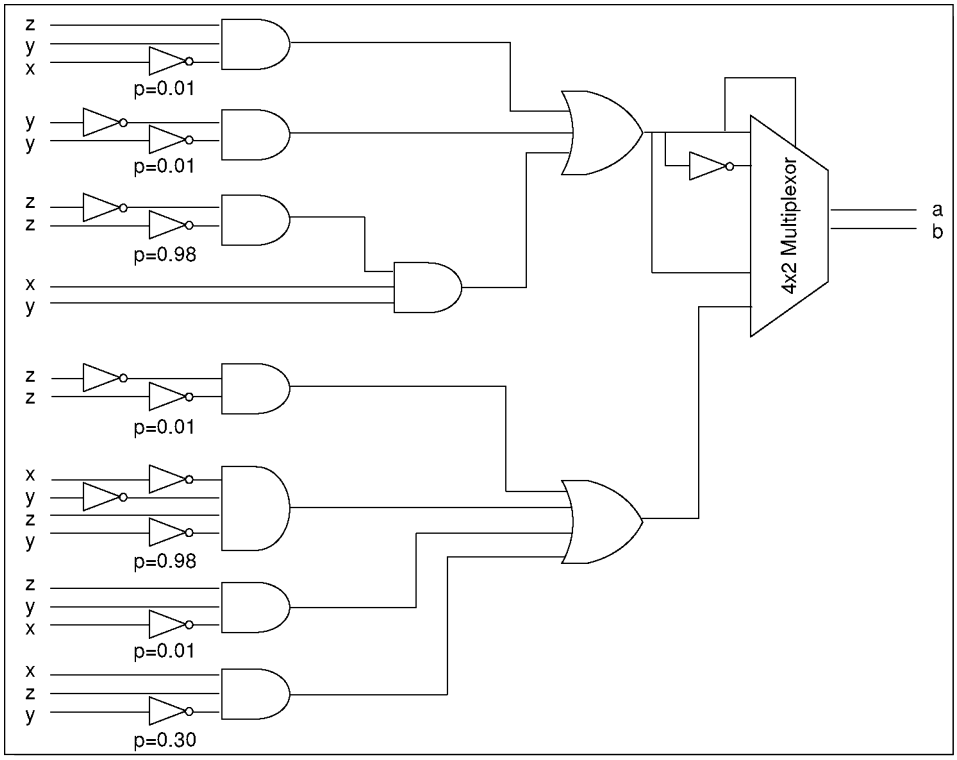


Fig. 11. The Boolean circuit which implements the inference step for one node in a Bayesian network with 37 nodes.

addition, since the probability parameter $p$ is controlled by varying the voltage, a high spread-factor implies that several distinct voltage levels are needed to operate the PCMOS devices in the chip. As pointed out elsewhere, supplying distinct voltage levels on a chip requires voltage regulators which are costly in terms of area as well as energy. We make two observations towards addressing this problem: (i) Distinct probability parameters are a requirement of the application, and the application's sensitivity to probability parameters is an important aspect to consider. In other words, if an application uses probability parameters $p_1$, $p_2$, $p_3$, it might be the case that the application-level quality of solution is not degraded much when only $p_1$, $p_2$ are used. However, this can be determined only experimentally. (ii) Given probability parameters $p_1$ and $p_2$, through logical operations, other probability parameters might be derived. For example, if the probability of obtaining the same a 1 from one PCMOS device is $p$ and the probability of obtaining the same from a second PCMOS device is $q$, a logical AND of the output of the two PCMOS devices produces a 1 with probability $p.q$. Using this technique, in the context of an application (the case of Bayesian inference is used here), the spread factor may be reduced by producing several probability parameters using a few probability parameters. The formulation of such an optimization problem is described next.

Consider a tree (the *composition tree*) with directed edges $G \equiv (V, E)$, where $V$ is a set of vertices and $E \subseteq (V \times V)$ a set of directed edges. This tree describes the set of logical operations required to derive the probability parameters required by an application.

Let $V = I \cup C \cup O$, where $I \cap C = C \cap O = I \cap O = \phi$ and $|O| = 1$. Let $I$ be the set of *input vertices*, $C$ the set of *computing vertices*, and $o \in O$ the *output vertex*. The input vertices are PCMOS devices, the computing vertices are logical operations, and the output of the output vertices yields the probability parameters required by the application. Given an edge $e \equiv (u, v) \in E$, where $u, v \in V$, the *value associated with the edge*, $val(e)$, is the value associated with the vertex $u$. In other words, $val(e) = val(u)$, where $e = (u, v)$.

—**Set** $I$: For any vertex $v \in I$, $val(v) \in \Re^+$.

—**Set** $O$: For the vertex $o \in O$, $val(v) = val(e)$ where $e$ is the incoming edge incident on $o$.

—**Set** $C$: Any vertex $v \in C$ is of one of three types AND, OR, NOT. For all vertices of type AND, OR, the in-degree is two and out-degree is one. For all vertices of type NOT, the in-degree is one and out-degree is one.

  —For any vertex $v \in C$ and $v$ of type AND with incoming edges $e_i, e_j$, the value associated with $v$, $val(v) = val(e_i) \times val(e_j)$.

  —For any vertex $v \in C$ and $v$ of type OR with incoming edges $e_i, e_j$, the value associated with $v$, $val(v) = 1 - (1 - val(e_i)) \times (1 - val(e_j))$.

  —For any vertex $v \in C$ and $v$ of type NOT with incoming edge $e_i$, the value associated with $v$, $val(v) = 1 - val(e_i)$.

Consider a set $P$ such that $P \equiv \{p_1, p_2, p_3, \ldots, p_k\}$, where $p_i \in \Re^+$ and $Q$ are such that $Q \equiv \{q_1, q_2, q_3, \ldots, q_l\}$, where $q_i \in \Re^+$. Let $P$ be called

Table V. Application-Required Probability Parameters and Composition Tree for Generating Them using Two Voltage Levels

| Application-Required Probability Parameters | Composition Tree |
|---|---|
| 0.05 | [[(0.4)AND(0.5)]AND(0.5)]AND(0.5) |
| 0.10 | [(0.4)AND(0.5)]AND(0.5) |
| 0.15 | [(0.5)AND[NOT(0.4)]]AND(0.5) |
| 0.20 | (0.4)AND(0.5) |
| 0.25 | (0.5)AND(0.5) |
| 0.30 | (0.5)AND[NOT(0.4)] |
| 0.35 | [NOT[(0.5)AND[NOT(0.4)]]AND0.5 |
| 0.40 | 0.40 |
| 0.45 | [NOT[[(0.4)AND(0.5)]AND(0.4)]] |
| 0.50 | 0.50 |

the set of *input probabilities* and $Q$ be called the set of *application-required probabilities*.

A composition tree $G_i$ is said to compute $q_i \in Q$ with input probabilities $P$ if for each input vertex $v$ of $G$, $val(v) \in P$ and the value of the output vertex of $G$, namely, $val(o) \approx q_i$, where $x \approx y$ if for some $\epsilon$, $y - \epsilon \leq x \leq y + \epsilon$. In other words, when elements from the set $P$ are input to the composition tree $G_i$, the value of the output vertex is $\approx q_i$.

For a composition tree $G_i$, which computes $q_i$ given $P$, $G_i$ is defined to be the minimal composition tree if $\nexists G_i'$ such that $G_i'$ computes $q_i$, so long as $P$ and the number of vertices in $G_i'$ is less than the number of vertices in $G_i$. Henceforth, a "composition tree" will refer to the minimal composition tree.

To compute the application-required probability parameters from a set of input probability parameters, the total cost includes the costs of PCMOS devices, that of the logic in the composition tree, and that introduced due to multiple (though reduced) probability parameters of the input.

The cost of computing $q_i$ which is given a set of input probabilities $P$, denoted by $C_P(q_i)$, is the number of vertices in composition tree $G_i$ which computes $q_i$, given $P$. The cost of computing the the set $Q$, given $P$, is denoted by $C_P(Q)$ and is $\Sigma_{q_i \in Q} C_P(q_i)$. The cost of the set of input probabilities, denoted by $\bar{C}_P$, is $\bar{C}_P = k \times |P|$, where $k$ is some constant.

*Question*. Given $Q$, compute $P$ and the composition trees such that $C_P + \bar{C}_P$ is minimum over all possible $P$.

This optimization problem might be solved using a combination of linear programming and heuristics. As an illustration, an (unoptimized) hand implementation for deriving 20 probability parameters from two input probability parameters is described in Table V (note that the other 10 probability parameters can be obtained by the NOT gates of those in the table).

### 6.4 Future Directions

For any implementation of applications which leverage probabilistic algorithms, its *quality* is an important aspect to consider, apart from the energy and running time. In conventional implementations of probabilistic algorithms—

which usually leverage hardware- or software-based implementations of pseudorandom number generators to supply pseudorandom bits which serve as coin tosses—it is a well-known fact that random bits of low quality affect application behavior, from the correctness of Monte Carlo simulations [Ferrenberg et al. 1992] to the strength of encryption schemes. To ensure that application behavior is not affected by low-quality randomization, the quality of random bits produced by a particular strategy should be assessed rigorously. The problem of quality assessment of random sequences has been well studied and is rooted in the very concept of "randomness". Kolmogorov considers a finite sequence to be random if there is no appreciably shorter sequence that describes it fully, in some unambiguous mathematical notation (from Good [1972]). However, the problem of determining the shortest sequence which describes a finite sequence of numbers is, in general, undecidable [Chaitin 1977]. A more practical definition of pseudorandomness was introduced by Yao, where, informally, a sequence is pseudorandom if there is no polynomial-time algorithm which can distinguish that sequence from a truly random one [Yao 1982]. However, it is impractical to test for pseudorandomness, since there is an infinite number of polynomial-time algorithms. Hence, the current strategy is to leverage statistical tests to test for the quality of randomness. To study the statistical properties of PCMOS devices in a preliminary way, we have utilized the randomness tests from the NIST suite [NIST 2007] to assess the quality of random bits generated by PCMOS devices. Preliminary results indicate that PCMOS affords a higher quality of randomization; a future direction of study is to quantify the impact of this quality on the application-level quality of solution.

A second direction of inquiry is automating design of PSOC architectures. As mentioned in Section 6.2, the description of the probabilistic steps of a probabilistic algorithm can be formalized through a probabilistic truth table. Given a conventional (deterministic) truth table, a corresponding Boolean circuit (of AND, OR, and NOT gates) can be created and minimized (e.g., the Karnaugh map technique [Mano 2001]) and such a synthesis and minimization step is automated in conventional CAD tools. A similar formalism and methodology for automating the synthesis of PSOC architectures based on probabilistic truth tables can be developed.

In general, we can consider three categories of applications: (i) applications which benefit from (or harness) probabilistic behavior at the device level; (ii) those that can tolerate probabilistic behavior at the device level; and (iii) those which cannot tolerate statistical behavior. In this article, we have described and evaluated our methodology for implementing the first category of applications. Moving away from applications that leverage probabilistic algorithms (and in turn harness the probabilistic behavior of PCMOS devices), we can visit the domain of applications that tolerate probabilistic behavior. Specifically, applications which can tradeoff energy and performance for the application-level quality of solution can be investigated. In this context, applications in the domain of digital signal processing are good candidates, where the application-level quality of solution is naturally expressed in the form of a signal-to-noise ratio. For the last category, we envision an approach based on redundancy, as well as error correction and recovery techniques.

## REFERENCES


BAHAR, R. I., MUNDY, J., AND CHEN, J. 2003. A probabilistic-based design methodology for nanoscale computation. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*. 480–486.

BEINLICH, I., SUERMONDT, G., CHAVEZ, R., AND COOPER, G. 1989. The ALARM monitoring system: A case study with two probabilistic inference techniques for belief networks. In *Proceedings of the 2nd European Conference on AI and Medicine*. 247–256.

BENNETT, C. H. 1973. Logical reversibility of computation. *IBM J. Res. Devel. 17*, 525–532.

BORKAR, S., KARNIK, T., NARENDRA, S., TSCHANZ, J., KESHAVARZI, A., AND DE, V. 2003. Parameter variations and impact on circuits and microarchitecture. In *Proceedings of the 40th Design Automation Conference*. 338–342.

CHAITIN, G. 1977. Algorithmic information theory. *IBM J. Res. Devel. 21*, 350–359.

CHAKRAPANI, L. N., AKGUL, B. E. S., CHEEMALAVAGU, S., KORKMAZ, P., PALEM, K. V., AND SESHASAYEE, B. 2006. Ultra efficient embedded SOC architectures based on probabilistic CMOS technology. In *Proceedings of the 9th Design Automation and Test in Europe (DATE)*. 1110–1115.

CHAKRAPANI, L. N., GYLLENHAAL, J., MEI W. HWU, W., MAHLKE, S. A., PALEM, K. V., AND RABBAH, R. M. 2005. Trimaran: An infrastructure for research in instruction-level parallelism. In *Proceedings of the 17th International Workshop on Languages and Compilers for Parallel Computing*. Lecture Notes in Computer Science, vol. 3602. Springer, 32–41.

CHEEMALAVAGU, S., KORKMAZ, P., AND PALEM, K. V. 2004. Ultra low-energy computing via probabilistic algorithms and devices: CMOS device primitives and the energy-probability relationship. In *Proceedings of the International Conference on Solid State Devices and Materials* (Tokyo, Japan), 402–403.

CHEEMALAVAGU, S., KORKMAZ, P., PALEM, K. V., AKGUL, B. E. S., AND CHAKRAPANI, L. N. 2005. A probabilistic CMOS switch and its realization by exploiting noise. In *Proceedings of the IFIP International Conference on Very Large Scale Integration*.

CORP., I. 1998. SA-1100 microprocessor technical reference manual.

DING, Y. Z. AND RABIN, M. O. 2002. Hyper-Encryption and everlasting security. In *Proceedings of the 19th Annual Symposium on Theoretical Aspects of Computer Science*. Lecture Notes In Computer Science, vol. 2285. Springer, 1–26.

DOBRUSHIN, R. L. AND ORTYUKOV, S. I. 1977a. Lower bound for the redundancy of self-correcting arrangements of unreliable functional elements. *Problems Inf. Transmis. 13*, 3, 59–65.

DOBRUSHIN, R. L. AND ORTYUKOV, S. I. 1977b. Upper bound on the redundancy of self-correcting arrangements of unreliable elements. *Problems Inf. Transmis. 13*, 3, 201–20.

FELLER, W. 1984. *An Introduction to Probability Theory and its Applications*. Wiley Eastern Limited.

FERRENBERG, A. M., LANDAU, D. P., AND WONG, Y. J. 1992. Monte Carlo simulations: Hidden errors from "good" random number generators. *Phys. Rev. Let 69*, 3382–3384.

FUKS, H. 2002. Non-Deterministic density classifiation with diffusive probabilistic cellular automata. *Phys. Rev. E, Statis. Nonlinear Soft Matter Phys. 66*.

GELENBE, E. 1989. Random neural networks with negative and positive signals and product form solution. *Neural Comput. 1*, 4, 502–511.

GELENBE, E. AND BATTY, F. 1992. Minimum graph covering with the random neural network model. In *Neural Networks: Advances and Applications*, vol. 2.

GEORGE, J., MARR, B., AKGUL, B. E. S., AND PALEM, K. 2006. Probabilistic arithmetic and energy efficient embedded signal processing. In *International Conference on Compilers, Architecture, and Synthesis for Embedded Systems (CASES)*.

TRIMARAN. 2007. Trimaran: An infrastructure for research in instruction-level parallelism. http://www.trimaran.org.

INTEL. 2007. Moore's law. http://www.intel.com/technology/silicon/mooreslaw/.

ITRS. 2002. International technology roadmap for semiconductors 2002 update.

JACOME, M., HE, C., DE VECIANA, G., AND BIJANSKY, S. 2004. Defect tolerant probabilistic design paradigm for nanotechnologies. In *Proceedings of the 41st Annual Conference on Design Automation*. 596–601.

KISH, L. B. 2002. End of Moore's law: thermal (noise) death of integration in micro and nano electronics. *Phys. Lett. A 305*, 144–149.

KORKMAZ, P., AKGUL, B. E. S., CHAKRAPANI, L. N., AND PALEM, K. V. 2006. Advocating noise as an agent for ultra low-energy computing: Probabilistic CMOS devices and their characteristics. *Japanese J. Appl. Phys. 45*, 4B (Apr.), 3307–3316.

LANDAUER, R. 1961. Irreversibility and heat generation in the computing process. *IBM J. Res. Devel. 3*, 183–191.

LEFF, H. AND REX, A., eds. 1990. *Maxwell's Demon: Entropy, Information, Computing*. Princeton University Press, Princeton, NJ.

MACKAY, D. 1992. Bayesian interpolation. *Neural Comput. 4*, 3.

MANO, M. M. 2001. *Digital Design*. Prentice Hall, Upper Saddle River, NJ.

MEINDL, J. AND DAVIS, J. 2000. The fundamental limit on binary switching energy for terascale integration (tsi). *IEEE J. Solid-State Circ. 35*, 10 (Oct.), 1515–1516.

NATORI, K. AND SANO, N. 1998. Scaling limit of digital circuits due to thermal noise. *J. Appl. Phys. 83*, 5019–5024.

NEPAL, K., BAHAR, R. I., MUNDY, J., PATTERSON, W. R., AND ZASLAVSKY, A. 2005. Designing logic circuits for probabilistic computation in the presence of noise. In *Proceedings of the 42nd Design Automation Conference*. 485–490.

PALEM, K. V. 2003a. Energy aware algorithm design via probabilistic computing: From algorithms and models to Moore's law and novel (semiconductor) devices. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (San Jose, CA), 113–117.

PALEM, K. V. 2003b. Proof as experiment: Probabilistic algorithms from a thermodynamic perspective. In *Proceedings of the International Symposium on Verification (Theory and Practice)* (Taormina, Sicily).

PALEM, K. V. 2005. Energy aware computing through probabilistic switching: A study of limits. *IEEE Trans. Comput. 54*, 9, 1123–1137.

PALEM, K. V., CHEEMALAVAGU, S., KORKMAZ, P., AND AKGUL, B. E. 2005. Probabilistic and introverted switching to conserve energy in a digital system. US Patent 20050240787.

PARK, S. AND MILLER, K. W. 1988. Random number generators: Good ones are hard to find. *Commun. ACM 31*.

PFEFFER, A. 2000. Probabilistic reasoning for complex systems. Ph.D. thesis, Stanford Univeristy.

PIPPENGER, N. 1985. On networks of noisy gates. In *Proceedings of the 26th Annual IEEE Symposim on Foundations of Computer Science*, 30–38.

PIPPENGER, N. 1989. Invariance of complexity measures for networks with unreliable gates. *J. ACM 36*, 531–539.

PIPPENGER, N., STAMOULIS, G. D., AND TSITSIKLIS, J. N. 1991. On a lower bound for the redundancy of reliable networks with noisy gates. *IEEE Trans. Inf. Theory 37*, 3, 639–643.

RABIN, M. O. 1976. Probabilistic algorithms. In *Algorithms and Complexity, New Directions and Recent Trends*, J. F. Traub, ed. 29–39.

NIST. 2007. Random number generation and testing. http://csrc.nist.gov/rng/.

SANO, N. 2000. Increasing importance of electronic thermal noise in sub-0.1mm Si-MOSFETs. *IEICE Trans. Electron. E83-C*, 1203–1211.

SIEWIOREK, D. P. AND SWARZ, R. S. 1998. *Reliable Computer Systems: Design and Evaluation*. AK Peters, Ltd.

SINHA, A. AND CHANDRAKASAN, A. 2001. JouleTrack: A web based tool for software energy profiling. In *Proceedings of the 38th Conference on Design Automation*. 220–225.

SOLOVAY, R. AND STRASSEN, V. 1977. A fast Monte-Carlo test for primality. *SIAM J. Comput.*.

VON NEUMANN, J. 1956. Probabilistic logics and the synthesis of reliable organizms from unreliable components. *Automata Studies*, 43–98.

WOLFRAM, S. 2002. *A New Kind of Science*. Wolfram Media.

YAO, A. 1982. Theory and application of trapdoor functions. In *Proceedings of the 23rd Symposium on the Foundations of Computer Science*, 80–91.

Received September 2006; revised March 2007; accepted March 2007